Classification of zebrafish embryo using various ML methods

Saoud Akram, Juliette Meurgey, Simon Zamora
Ecole Polytechnique Fédéral de Lausanne, Switerland

I. Introduction

The formation of the precursors of the segmented vertebral column along the body-axis of a vertebrate embryo is governed by a complex genetic machinery. Those precursors, so-called somites, appear in a rhythmic and sequenced manner during the development of an embryo[1]. Their formation depends on the precise oscillation of a complex genetic network termed the segmentation clock. This clock allows a precise space-time formation of somites along the body axis of the embryo. A lot of work is being done to understand how such complex structures can appear under the supervision of the segmentation clock and what are the underlying mechanism.

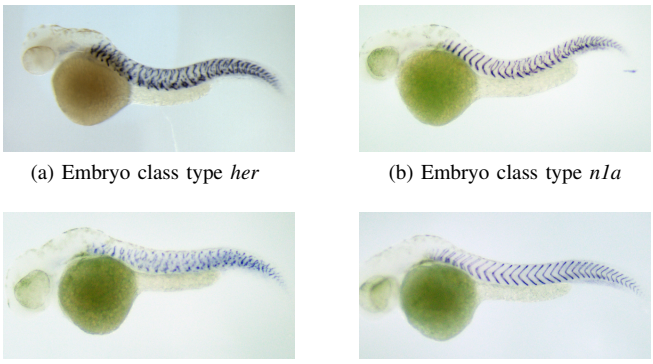
One classic model organism that is used to study the development and formation of the diverse structure of an embryo is the Zebrafish, (*Danio Rerio*). It has the advantages of growing externally after fertilization and thus can be observed easily throughout its whole development. It presents a perfect model to study the formation of somites under the control of the segmentation clock as it can be genetically manipulated. The effect of a mutation on specific genes can thus be directly monitored by observation of their phenotype.

This project was done in collaboration with the *Segmentation Timing and Oscillation laboratory*, directed by professor Andrew Oates at EPFL. The main objective of the project was to create a classifier that could classify images of embryos depending on the specific phenotype they showed. For this purpose, the lab sent us images of embryos of Zebrafish that presented four specific phenotypes corresponding to their four different genotypes. We had at our disposal three different mutant type embryos: Tbx6 mutants, the so-called fussed somite, that showed a phenotype characterized by a complete lack of somite formation along the body axis.

Herst mutants, that present mutations in two different genes Her1 and Her7. These are oscillatory genes that are important for the segmentation clock, so any perturbation in their functionality will give rise to defective, not well-defined somites along the whole body axis.

N1a mutant, also called Notch 1a, corresponds to a mutation on a gene that takes part in local cell-to-cell signaling which is essential for precise synchronisation of the cellular oscillation. Mutations in this gene result in a disorganised oscillation of the whole system and result in somite defects appearing progressively along the body axis.

Finally, the last type of embryos we had at our disposal was wild-type embryos that presented no defects in their somite formation. Having this image set at our disposal we proceeded to create classifiers that could correctly classify each image to their corresponding mutant classes using various machine learning techniques.



(a) Embryo class type *her* (b) Embryo class type *n1a*

(c) Embryo class type *tbx6* (d) Embryo class type *wildtype*

Figure 1. The four provided embryo class types

II. Problem Statement

Our main objective was to build a reliable classifier for the four different types of zebrafish embryo in our data set : *her, n1a, tbx6 and wildtype*. Once we had this classifier, our goal would be to try to gain insight into which features it used to classify each embryo by creating the activation map of each neural network model. In this way, we could hopefully extend the capacity for humans to classify zebrafish embryos with methods they would not have considered otherwise.

III. Initial Data Cleaning

The initial data set contained 509 specimens : 78 *her* mutant, 161 *n1a* mutant, 57 *tbx6* mutant and 213 *wildtype*. The images differed somewhat from each other; the framing was centered differently from one to the next, there were sometimes hairs or drops on the image, and the relative orientation of the embryo from the framework. For these reasons, we decided to crop the images by hand and we also flipped some of the images so that that head of the embryo was on the left side to have a standardized input. We used the Crop function from ImageJ to crop each image to the size of 544 by 160 pixels. We then saved the images with a Tiff format to conserve the original data format.

Figure 2. Cropped image of size 544 x 160

IV. Programming environment & Libraries used

Our entire project was done in Python 3.7. Given the expensive computational cost that arises from the use of Deep Neural Networks to classify images, we used Google Colab as our hosting environment and ran our notebooks on the provided GPU. This was the hosting environment that we used for our final version of our code.

The libraries we used for this project are:

- Scikit-learn, which we mainly used for the KNN algorithm as well as to print our Confusion Matrix
- OpenCV, that we used for image processing
- Keras, which we used ro deploy, train, and evaluate our Convolutional Neural Networks

V. Methods Used & Results

A. K-NN approach

Our first attempt at classifying these images was to use the K-nearest neighbours (K-NN) algorithm. We devised two models, with different input vectors. The first one used all the pixels of the input image and then transformed it into a 1-dimensional vector. The second model created a 1-dimensional array of the RGB histogram of the input image. This allowed us to have a baseline in regards to the classification accuracy which will serve us when assessing the quality of subsequent and more complex models. To train both models and find the best k, we used hold-out cross validation (i.e. we split our training set into a training and a validation set and then used the validation set to find the best k that classifies our data).

The accuracy of the predictions generated by both models ranges from 60% to 75%. This is mainly due to the fact that the K-NN algorithm is very sensitive to the relative position of each pixel so it is not much of a surprise for us that this algorithm doesn't perform well when taking into consideration the high variability of embryos positions in our data set. The confusion matrix shows us that the first model can easily classify *n1a* mutants (often reaching a precision of 75% or slightly higher) while the precision for *her* mutants is quite low (50% or below). For the second model, we observed that it could more easily classify images of *wildtypes* (reaching a precision of approximately 80%) and had decent results with *n1a* and *tbx6* mutants (precision higher than 65%), but, again, isn't efficient with *her* mutants. The fact that both these models struggle with classifying *her* mutants is due to the fact that we are using an unbalanced dataset containing a small number of images (508 in total).

B. Data preparation for Convolutional Neural Netwrok

Before beginning analysis on the data set using Convolutional Neural Network, we had to prepare what would become our Train set and our Test set. For this purpose, we separated our complete data set in two. We randomly took one-fifth of our data set for our test set (102 images) and the rest for our train set (407 images). The test set was used to evaluate the performance of our model as the train set was used to fit our model. This way we could assess the precision of our model on a data set it has not seen, thus reflecting its real performance. For the rest of the report, we used the term accuracy to refer to the performance of the model on the test set.

C. Basic Convolutional Neural Network approach

After having established a baseline of prediction precision we wanted to start using a Convolutional Neural Network as they are more adapted for an image classification task. We used the *Keras* libraries to build a *Sequential model*. We built our own CNN with the following simple architecture:

Layer Type	activation function
Convolutional2D (32 filters)	relu
MaxPooling2D layer	
Convolutional2D (64 filters)	relu
MaxPooling2D layer	
Dropout Layer 0.4	
Flatten Layer	
Dense layer (4 nodes)	softmax

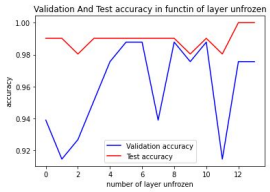
Architecture of the Convolutional Neural Network

The Dropout layer was added to avoid overfitting on our training set. The factor 0.4 next to it means that 40% of the layer will be dropped to construct the final model. The whole model is relatively simple regarding more complex CNN and had a total of 1'412'036 trainable parameter. Once trained on our trained set the model achieved a precision of 68% on the test set (missclassifying 34 out of 102 images). We didn't try to fine-tune this model as we knew that we wanted to move on to pre-trained models that would certainly give us a higher precision.

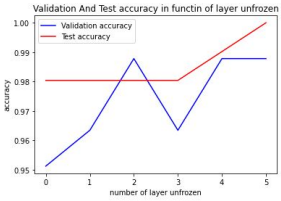
D. Transfer Learning - VGG16 and MobileNetV2

Given the fact that our dataset is very small, it was natural for us to look towards pre-trained Convolutional Neural Networks (CNN) in order to classify our images. We used these pre-trained models as starting points for our own models, which we fine-tuned to our task at hand. This process is called Transfer Learning.

The first pre-trained model we considered is VGG16 which has approximately 138 million pre-trained parameters.



(a) Validation and test accuracy MobileNetV2



(b) Validation and test accuracy VGG16

Figure 3. Test and validation accuracy in function of layer unfrozen

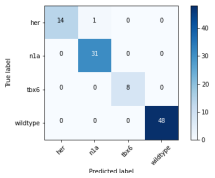


Figure 4. Confusion Matrix for MobileNetV2

We changed the input layer such that the model accepts 544x160 images. We also added a Dense layer (which is there for combining features that the VGG16 model has recognized in the image) as well as a Dropout layer (which may prevent overfitting and, as such, improve generalization) before adding an output layer that sorts the pictures in one of four classes. The second pre-trained model we used was MobileNetV2 a lighter model with about 4.3 million pre-trained parameters. Just like our model based on VGG16, we also changed its input layer for it to accept our own images and added this time only one output layer to allow the multi-class classification needed. When training, we used Categorical Cross-Entropy Loss, as it is the one that is usually used when undertaking multi-class classification. There were other parameters that we used for the compilation of both our models. We used as an optimizer the *Keras* Adam optimizer with a learning rate of 0.0001 and to evaluate the performance of our model we used the accuracy metrics provided by *Keras*. For the moment we only allowed training on the layer that we manually added and froze the rest of the model.

The test accuracy we got by building a model for each of those pre-trained CNNs was very high, with VGG16 achieving an accuracy of 97% (with only 3 out of 102 images misclassified) and MobileNetV2 achieving 93% (with 7 out of 102 images misclassified).

It is important to mention that both VGG16 and MobileNetV2 were initially trained on the ImageNet dataset, which includes more than 14 million images belonging to 1000 classes but none having anything close to a mutant zebrafish embryo. Furthermore, both VGG16 and MobileNetV2 have been trained using images of a different square format, 244x244 for example, so we found it really interesting and encouraging to see that without any fine-tuning the pre-trained model reached such good precision.

E. Best Model Selection

When building a model of your own off a pre-existing one, there are multiple parameters to tune such that the model best fits the problem at hand. First of all, we need to consider how many layers of each pre-trained model we will ”unfreeze” (i.e. we look at how many of these pre-trained layers we need to re-train to try and optimize the validation accuracy and validation loss). In order to achieve this, we trained a model whilst unfreezing a certain number of layers and chose the model which had the lowest loss in the validation set. To compute this validation loss, we used the *validation_split* functionality from *Keras* that generates a validation set automatically at each new epochs. We chose a value of 0.2 for this parameter meaning that at each step, a randomly chosen fifth of the total training set will be used as a validation set.

When training, to find the optimal number of epochs, we followed an ”earlystop” methodology to find the optimal number of epochs to use. This method checks if the last registered validation loss for a given epoch e is the lowest across the next 6 epochs (if it is, then it stops and reverts to the weights learned after e and stops the training ; if it isn't, it sets the lower validation loss as the last registered validation loss and restarts). To further optimize the VGG16 model, we tried to find the optimal values for the number of hidden nodes as well as the batch size.

Figure 3. shows the resulting evolution of the test and validation accuracy in term of the number of unfrozen layer when the model was trained with the best hyper parameters. We can see that for MobileNetV2, the maximum of the validation accuracy (in blue) reaches its maximum at 9 unfrozen layer and with a batche size of 10. The model manage to achieve a precision of 99% for the test set. A higher precision on the test set has been achieved with a higher number of layer unfrozen but we can see that the validation accuracy decrease meaning that the overall precision of the model was poorer. VGG16 has the maximum validation accuracy with 5 unfrozen layer (and 512 hidden layers) and manage to correctly predict all the images of the test set. In Figure 4. you can see the resulting confusion matrix of MobileNetV2 on our test set. In this case our model manage to classify all images except one her mutant that was classified as n1a.

F. Data Augmentation - Testing Robustness

In order to test the robustness of our models, we augmented our test set to assess its performance. As MobileNetV2 perform really poorly we only report the result of VGG16. It is understable knowing that as MobileNetV2 has 4 times less parameters than VGG16, it present a much lower robustness. We also trained our VGG16 model with

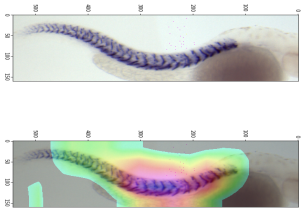


Figure 5. Example of an activation map for one image (type *her*)

augmented training sets to see how they would perform on non-augmented sets and augmented sets. As such, we will consider 4 forms of data augmentation:

1) Non-Augmented (A)
2) Flips and Rotations (B)
3) Zooming and adding noise (C)
4) Flips, Rotation, Zoom, Noise (D)

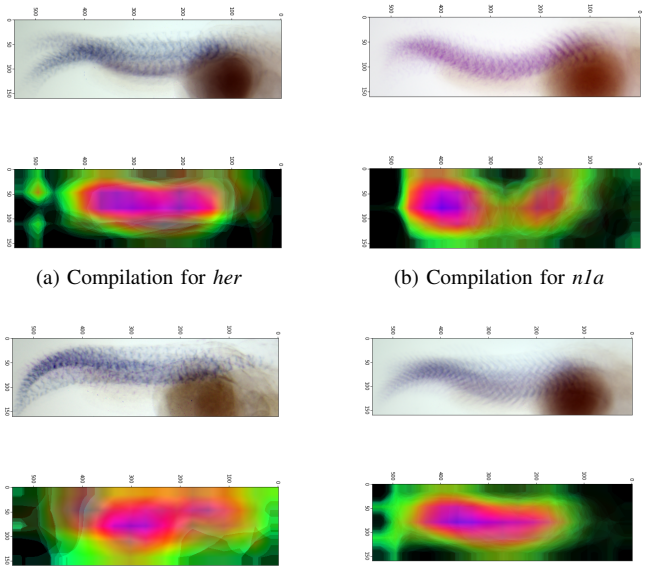
For each augmented dataset, we trained our model and looked at the efficiency on augmented and non-augmented test sets. Here are the resultin accuracy on the test set we got:

-	A	B	C	D
A	0.951	0.76	0.56	0.54
B	0.99	0.98	0.52	0.66
C	0.94	0.71	0.84	0.71
D	0.95	0.96	0.9	0.9

This table can be understand this way. AA correspond to the test accuracy when neither the training or the test set was augmented. CB corresponds to the accuracy when the training set was augmented using Zooming and Noising and the test set was augmented using Flips and Rotation (in other words, rows correspond to the training sets and columns to the test sets). These results are quite interesting. It makes sense that when taking more and more diverse pictures when training, the model is able to generalize (which explains why row D seems to have the best accuracies, while row A seem to have the worst).

G. Class Activation Maps

After looking at how robust our dataset was on transformed images, we looked at which parts/pixels of each image in the test set have contributed more to the final classification output of our models. This could be of great use for the labs as it could indicate the presence of patterns in the embryo which is specific to each mutant class. Also, for our problem we expect the model to be activated by the embryo's dorsal somites as it is there that all the various mutation on the genome produce the different phenotype. To produce those activation map, we extracted the last Convolution layer in our MobileNetV2 model and used them to create a heatmap[2]. We then superimposed all heatmap onto a superimposed set of all images for each class to see



(a) Compilation for *her* (b) Compilation for *n1a*

(c) Compilation for *tbx6* (d) Compilation for *wildtype*

Figure 6. Compilation of class activation maps for the four types

the most important features. We produce the resulting Class Activation Map (CAM) for each of our mutant type which is presented in Figure 6.. By plotting these superimposed images we could see which part of the zebrafish embryo image carried the most information for classification from the point of view of the model.Despite the high variability in the embryos position from one image to the other, from which result the blurriness of the top compilation, the CAM manage to highlight some region on the dorsal spine of the embryo. What will be interesting for the lab is the variability of CAM highlight from one class to the other. For example, the position of the bottom of the spine seemed to be a more determining factor than the top in the identification of the *n1a* type. We gave all those resulting CAM to Oates Lab for further analysis.

VI. Conclusion

We were able to produce a zebrafish embryo classifier with a high accuracy based on the images we were provided. Furthermore, in the generation of the class activation maps, we were able to determine the regions of the images that were the most useful in the classification. We ran our classifier on test images and obtained an accuracy of 99%. This goes to prove that our model is useful to classify images that were not used to fit it the model and thus could serve future researchers in the classification of other zebrafish. A future goal of this project would be to delve deeper into the class activation maps to see if we could identify classification features with more detail, such as individual vertebrae rather than just areas of the image.

ACKNOWLEDGEMENTS

Many thanks to Martin Weigert, Arianne Bercowsky Rama, and Professor Andrew Oates from the Segmentation Timing and Dynamics Laboratory, EPFL, Switzerland

REFERENCES

[1] A. C. Oates, L. G. Morelli, and S. Ares, "Patterning embryos with oscillations: structure, function and dynamics of the vertebrate segmentation clock," *Development*, vol. 139, no. 4, pp. 625–639, 2012. [Online]. Available: https://dev.biologists.org/content/139/4/625

[2] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2921–2929.